A GRAPHICS-APPLICATIONS DATA STRUCTURE FOR MEDIUM SCALE COMPUTERS

by

Luke Horrell Miller

United States
Naval Postgraduate School

THESIS

A GRAPHICS-APPLICATIONS DATA STRUCTURE
FOR
MEDIUM SCALE COMPUTERS

by

Luke Horrell Miller, Jr.

December 1970

A Graphics-Applications Data Structure for Medium Scale Computers

by

Luke Horrell Miller, Jr.
Lieutenant, United States Navy
B.S., University of Texas, 1963

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 1970

ABSTRACT

This paper documents the process by which a data structure for a Computer-Aided-Design system was selected and implemented on a medium scale computer. Included is a survey of the types of structures currently used in C.A.D. applications and a discussion of their capabilities and resource requirements.

TABLE OF CONTENTS

I. INTRODUCTION

The metamorphosis of a computer with graphic input-output capability into a general computer-aided design system is dependent primarily upon the development of a scheme for modeling the problems to be solved. Such a model, or data structure, must be capable of both creating the display file for graphical representation of the problem, and presenting the application programs with data in a useable form.

Terms that will be used to discuss data structures are:

1) Pointer - the machine address of a Data Cell;

2) Data Item - one or more machine words containing data;

3) Data Cell - a contiguous block of memory containing Data Items;

4) Low Level Data Structure - a modeling scheme in which the mechanisms for construction of the structure are defined by the user;

5) Pre-defined Data Structure - a modeling scheme in which the mechanisms for construction of the structurn are pre-defined.

The selection of the data structure to be incorporated within a CAD system is at best a complex process. No established procedure

exists by which firm conclusions can be drawn about the advantages and disadvantages of the various modeling schemes available.

The simplest structure with which to build a problem model is one in which the structure is pre-defined (i.e., a high level structure). In the pre-defined structure the data structure language is used to describe and manipulate the ordering of cells within a fixed structure. The ease of use is achieved by sacrificing flexibility. Not all problems can be efficiently modeled by a fixed structure, hence selection of such a structure is de facto a decision to reduce the systems capability in certain design problem areas.

Somewhat greater flexibility is provided by the low level data structures. In these only the data cells and the method of expressing relationships between them are pre-defined. The associated language enables the user to build from these cells, using the prescribed relational mechanisms, structures of the various forms. L^6 [Ref. 1] is an example of this type of structuring method. Within limits the user is capable of tailoring the structure to more closely meet the needs of the problem to be solved.

On a lower level still are the generalized data structures such as AED [Ref. 2] which have the capability to model any problem that can be formalized. The power of these modeling schemes arises from the ability of the user to define both the data cell and the methods for expressing relationships between them. Using a variety of techniques (i.e., hash coding, dictionary lookup, stacks, rings, etc.),

structures of any type and degree of complexity can be constructed. By careful analysis of the design problem, a structure can be built which provides an optimized model. When a CAD system is expected to be applied to a greatly diverse problem set, the implementation of a generalized structure is virtually mandatory. Unfortunately the necessity to define the structure and all the relationships within that structure before work can progress towards solution of the problem places a large additional burden on the system user.

Computer-aided design systems have, in general, been developed on computers with large main memories. The selection of which philosophy of structure (pre-defined or generalized) and which specific structure within that philosophy to implement on such a computer may be based primarily on the nature of the problems to which the system is to be applied and the skill of the users. When large amounts of main memory are not available, as in the case of medium sized computers, then the selection of data structure may be strongly influenced by the computer resources available.

The choice of a structure to be used by a medium sized computer should incorporate the best possible trade off between efficiency of memory utilization and processor time while placing as few restrictions as possible on the size and complexity of the models to be created.

The purpose of this paper is to document the process by which a data structure was selected and implemented on a medium sized

computer with intelligent graphic terminals. The particular equipment involved in the implementation was an XDS-9300 with AGT-10 graphic terminals. While many of the specifics of the implementation are machine dependent the criteria of selection and general techniques of implementation have application to other equipments of similar capability.

II. SELECTION OF A DATA STRUCTURE

There exists a variety of data structures which either have been developed expressly for use in computer-aided-design systems or lend themselves to that application. All, in general, share the property of explicitly defined relationships between data items on both a commonality and hierarchical basis. Gray [Ref. 3] noted that additionally CAD structures should be capable of dynamic growth and change as well as have the ability to be entered at any point within the structure. Although the form and mechanisms of such structures vary, most can be classified into distinctive groups.

The first group is that of Set-Theoretic data structures. Childs [Ref. 4] has developed a system by which data is stored contiguously without pointers. A block of pointers to the data items is created (the Beta block) each with a unique integer associated with it. A second block (the Eta block) contains sets of integers, each set then defining a relationship between data items. Mechanisms for defining sets and performing set operations complete the system and provide the capability of defining completely arbitrary relationships in any structure desired. Other systems based on set-theory operations have been developed and while offering great potential in CAD applications, have not achieved wide usage. This is possibly due to the difficulty in

implementation of such a system and the relatively early stage of development.

The second grouping of structure schemes contains those based on associative memory techniques, of which the LEAP data structure by Rovner and Feldman [Ref. 5] is perhaps the best example. LEAP operates on triples, where the triple is an ordered set of three data items - ATTRIBUTE, OBJECT, and VALUE. Retrievals are made on the basis of all three data items, any two or any single item. This is accomplished by means of three separate structure spaces: Attribute space; Object space; and Value space, with a copy of the triple in each space. Each triple is stored as a permutation in which the first data item corresponds to the name of the structure space. Within each structure space the permutation is stored at an address determined by hash-coding two of the three data items. Retrievals of the form ATTRIBUTE OF (UNSPECIFIED) IS VALUE can be made by accessing the attribute structure space, hashing the attribute and value data items together and retrieving the desired information from the resulting address. The Attribute structure space contains two registers each "containing" a unique ATTRIBUTE value (e. g. "father") and a pointer to a chain of triples in the structure space with the same unique ATTRIBUTE. In this fashion retrievals of the form ATTRIBUTE OF (UNSPECIFIED) IS (UNSPECIFIED) may be made. The Object and Value structure spaces contain similar registers.

LEAP also contains provision for inserting data items into unordered sets, creating rings of items with the same ATTRIBUTES.

LEAP and similar systems, due to the duplication of storage, require large amounts of storage for relatively simple models, but in general offer good search efficiency.

The third general category of data structures contains those based on the concept of "rings". The large majority of structures which have found application in CAD systems are elements of this classification. Included here are the structures developed with a view toward graphic applications, such as the rings of Sutherland's SKETCHPAD [Ref. 6], CORAL [Ref. 7], ASSOCIATIVE STRUCTURE PACKAGE (ASP) [Ref. 8], etc., as well as some of the more general list processing structures exemplified by SLIP [Ref. 9].

Ring structures are characterized by a basic structure consisting of a ring start which contains a pointer to the first data cell in the ring. Each cell "points" to the next, and the last cell "points" back to the ring start. Rings may be two-way linked with both forward and backward pointers. A data cell may be variable in length (i. e. contain a variable number of data items) and in general can be a member of more than one ring. Hierarchy may be denoted by placing rings on rings on rings, etc.

Ring structures may require less processor time for some operations than other structures since the addresses of data cells

required for retrieval or structure manipulation operations are stored with the data cells rather than computed. The decrease in processor time is of course traded off against the increased storage requirements.

In examining the various structure groupings from the pre-defined vs. generalized standpoint a few conclusions can be drawn. Set theoretic techniques seem to offer the greatest potential for generalized structures. Set theory can be considered a form of "relational calculus" and provides a formalized method of defining and operating upon relationships. Childs' STDS would appear to satisfy all the requirements of a generalized structure. Associative memories also appear to be fertile ground, particularly in hardware form as opposed to software simulated. LEAP, as Hamilton [Ref. 10] points out, also contains generalized structural ability. Ring-based structures are as a rule of the pre-defined type.

AED previously described as a generalized structure does not fall readily into any of the three groupings, principally because its structure is so user defined.

In the selection process of a data structure for the medium scale computer, the first two groups were eliminated from consideration. The set-theoretic structures were discarded primarily due to the problem of constructing an ordering scheme for the sets of data items. A set theoretic data structure is dependent upon all sets being well ordered and that ordering being preserved under union. If all sets

are well ordered then set operations can be accomplished by a merge or binary search. Lacking well ordering of sets, the execution of set operations becomes inefficient and the system impractical. The user of a set theoretic structure must define the well ordering relationships of the sets involved. Each set must have a unique linear representation which reflects the rank and preserves the order of its elements. The construct of a mechanism for such set representations which covers the wide range of data items operated upon in a CAD environment is a decidedly non-trivial problem.

Associative memory structures were not seriously considered because of their rather excessive storage requirements. The LEAP structure's speed of operation is relatively independent of structure size due in part to its efficient use of secondary storage. LEAP appears to offer great potential when dealing with very large problems, but on medium-sized computers the storage requirements of associative techniques appear prohibitive. The XDS-9300 at N. P. S. is configured with 32K of main memory and approximately 250K available to the user on secondary storage. The present monitor system provides no random access file capability hence secondary storage (drum) is organized sequentially. The already high processor overhead of associative techniques coupled with the large additional burden of paging in a sequential environment would appear to add strong further argument against implementing such a structure on the 9300.

AED was not selected for two reasons. A large amount of software is associated with the implementation of AED. This software includes the AED Jr. language specifier, the AED Processor, and many system packages which provide the mechanisms for manipulating the structures defined. This software, which must reside on secondary storage in its entirety and in main memory in part (at execution time), effectively reduces the storage space available for structures and applications programs. This reduction becomes important when main memory is small as secondary storage will undoubtedly be required for overlay of large applications programs or for paging of large data structures. The large software requirements of AED are, of course, related directly to its generalized nature. The limited scope of the problems expected to be run on the 9300 did not appear to justify a generalized system.

The second reason AED was removed from consideration was the difficulty of maintaining an interface between it and application programs. As mentioned in the discussion on generalized structures the user is faced with the responsibility of completely defining the data structure used to model the problem. Application programs are, in a sense, partially defined by the data structure. For each unique structure defined by the user a unique application program must be written to access that structure. This adds further to the already heavy burden this modeling system places on the user.

L[6] and other low level structures were eliminated from consideration on the basis that the principle users (engineers) would more effectively be able to use the system if they were freed from the task of defining the data structure. The choice was thus narrowed to the group of ring based structures. The structures in this group differed primarily in implementation, and the final selection was made on that basis.

The selection of languages available on the N. P. S. XDS-9300 is limited to FORTRAN IV and assembly language. Any implementation of a data structure on this machine must be either embedded in FORTRAN or implemented in the form of a high level data structure language with its associated processor. The latter method requires the user to learn a new language in which to define his models. A data structure language generally does not have the computational power of Fortran, and it would not be able to utilize the existing supply of Fortran application programs. Another factor is the probability of a data structure language processor requiring more secondary storage than the macros or subprograms required to embed a data structure in an existing high level language. For these reasons a structure compatable with Fortran was considered to offer the best utilization of the available resources.

SLIP developed by J. Weizenbaum is a list processing system based on ring structures very similar in concept to the rings of SKETCHPAD and CORAL. Weizenbaum refers to the rings as

"symmetric lists". Although SLIP is language and machine independent in concept, Weizenbaum's original implementation was in Fortran, and Ref. 9 describes it in that form. This demonstrable compatability with Fortran coupled with the resulting ease of use by 9300 users made it a reasonable choice for implementation.

III. IMPLEMENTATION OF SLIP

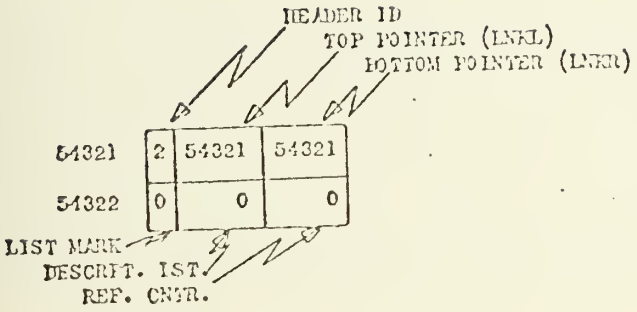
Some basic definitions must be made in order to clearly describe Weisenbaum's data structure system. The data cell contains two data items. The first contains three fields: the ID field; the LNKL (left link) field; and the LNKR (right link field. The function of the ID field is to provide storage for an integer which acts as a descriptor for the data cell utilization. There are four utilization codes which are discussed below. The LNKL and LNKR fields contain pointers to the next left and next right cells respectively in the ring. The second data item in the cell can be considered to contain pure data (i. e. non-specified) but may be subdivided into fields in the same manner as the first data item. Figure 1 illustrates the format of the SLIP cell.

A ring originates with a HEADER cell. This cell carries an ID code of 2, the LNKL points to the last (left most from the standpoint of the HEADER) cell in the ring and the LNKR points to the first cell in the ring (See figure 2). The second data item in the HEADER contains three fields: the LIST MARK field; the DESCRIPTION LIST POINTER; and the REFERENCE COUNTER. The LIST MARK field contains a user defined integer code allowing the user to mark lists for any desired purpose. The DESCRIPTION LIST POINTER, when non-zero points to an attribute list which "describes" the list headed

by the HEADER. The REFERENCE COUNTER contains the number of lists of which the list headed by the HEADER is a sublist. This number is important in the storage management process which is described below.

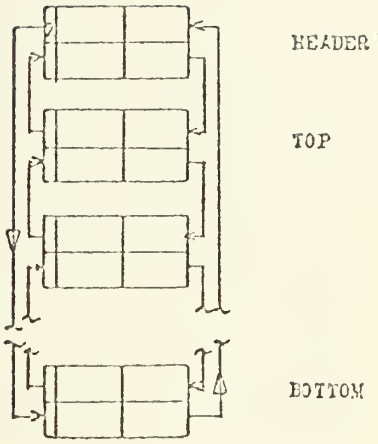
A HEADER is referenced by its NAME. A NAME is a data item in which the ID code is zero and the LNKL and LNKR fields both contain pointers to the HEADER. A HEADER may have its NAME stored in many locations. The designations of these locations are known as ALIASES for the NAME of the HEADER.

The ring structures of SLIP are accessed primarily by means of READER cells (See figure 3). A READER has an ID code of 3, the LNKL field initially points to the HEADER of the ring the READER is to "read", and the LNKR field is zero. Should the structure to be "read" contain sub-rings, and it is desired to read the sub-rings, then a READER is appointed for each sub-ring as encountered and the previous READER is pushed down on a READER stack. The LNKR field of the READER cell is used to maintain the chaining of the READER stack. The LNKL field of the READER is set to point to the next cell to be read after each read operation. The second data item of the READER cell contains a zero in the first field, a pointer to the HEADER of the ring being read in the second field, and a LEVEL COUNTER in the third field. The purpose of the pointer to the HEADER (which is not modified as the structure is read) is to enable the READER to be re-initialized without having to read the entire ring.



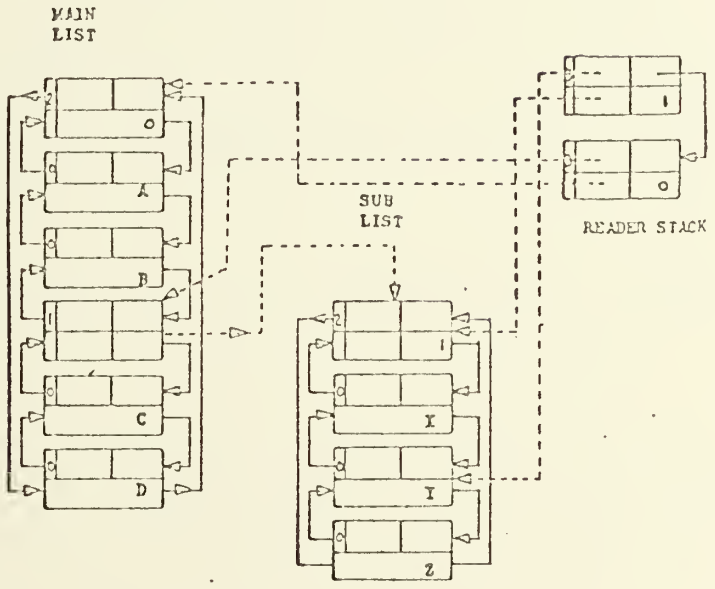
An Empty List

FIG. 1



A Simple List

FIG. 2



(A, B, (X, Y, Z), C, D)

A Simple List Structure

The READER is shown pointing to the cell containing Y.

FIG. 3

The above figures were extracted from Ref. 9

The LEVEL COUNTER indicates the number of READER cells pushed down on the READER stack (i. e. how far down the current READER is in the hierarchy of the structure).

The SLIP structure is organized around the basic ring which consists of a HEADER followed by an arbitrary (including zero) number of ring cells. A null ring consists of a HEADER whose LNKL and LNKR fields point to the HEADER. The ring cells carry on ID code of 0 or 1 and the LNKL and LNKR fields chain the ring cells together. The second data item in the ring cells (henceforth referred to as a DATUM) contains either information or a NAME as indicated by the cell ID code. A cell ID of 0 indicates the contents of the DATUM is data in user specified form. If the ID is 1, then the DATUM contains a NAME (points to a ring HEADER) and indicates that the ring with that HEADER is a sub-ring. Note that a ring may be a sub-ring of more than one ring and in fact may be a sub-ring of itself. There is no limit to the levels of hierarchy that can be defined.

Since rather elaborate and complex ring structures can be constructed with SLIP, a sophisticated system of access is required. Weizenbaum has incorporated into SLIP a scheme by which the structures created can be traversed in any order desired. This traversal is accomplished by means of the advance functions and READER cells. Advances are of two general types--structural and linear. In the structural advances whenever a NAME is encountered as a DATUM a new READER is appointed, the old READER stacked,

and the advance continued in the sub-ring pointed to by the NAME. The linear advance functions do not allow the advance operation to ascend or descend in the structure hierarchy. Advances can be made either clockwise or counter-clockwise in the ring and with one of three termination criteria. Advances are specified to be on the basis of Word, NAME, or Element. An element is a data cell not containing a NAME, and a Word is the union of the classes NAME and Element. HEADER cells are categorized separately and advances for a given level of hierarchy terminate on reaching a HEADER cell. The structural advance "pops" the READER stack and continues the search on the next higher level. If the termination criterion is Word, the linear advance operation accesses the next data cell in the ring and returns the DATUM of that data cell if that cell is a Word (i. e. not a HEADER). Should the DATUM of the cell pointed to by the READER when the advance function is called contain a NAME, then the structural advance would return the DATUM of the first Data Cell of the ring pointed to by the NAME. In the case of a linear advance based on NAME, the advance continues cell by cell until a cell with ID code 1 is reached. At that point the advance halts and the NAME is returned as the function value. The structural advance operates in the same fashion with the following exception. If the READER is pointing at a Data Cell containing a NAME when the advance function is called, then the ring pointed to by that NAME is accessed and the first NAME

encountered there is returned. An entire structure may be traversed if a structural advance on the basis of Word is called for and the advance is initiated at hierarchy level zero.

Storage management in SLIP is accomplished by means of maintaining a list of available storage (AVSL). The AVSL contains all unallocated data cells chained together in a one-way linked list. As cells are required they are allocated from the top of the list and when cells are deleted from the structure they are returned to the bottom of the list. When all references to a ring have been deleted that ring is also deleted (permanent rings can be created by the user which will not be automatically deleted if desired). This feature is made possible by the REFERENCE COUNTER in the ring HEADER. Each time a cell is created whose DATUM contains a NAME, the REFERENCE COUNTER of the ring HEADER pointed to by the NAME is incremented. When that cell is deleted the REFERENCE COUNTER is decremented. Should the counter become zero, then the ring is returned to the list of available storage.

SLIP provides the user with a full range of mechanisms for placing information on rings and manipulating those rings. Rings may be split or merged. Cells may be added or deleted at any point within a ring. The contents of any cell may be readily accessed or altered. For a full description of these and the other mechanisms of SLIP, the attention of the reader is invited to Ref. 9.

Weizenbaum's Fortran implementation of SLIP is comprised of a set of Fortran subprograms which are designed to be machine independent and a set of primitive assembly language subprograms. These highly machine dependent routines provide the interface between SLIP and the computer on which it is to be utilized. The primitive routines define the data cell, establish and retrieve the contents of the data cells and provide the recursive capability required by the Fortran subprograms.

The implementation of SLIP on the XDS-9300 involved primarily the creating of the primitive routines. First, however, a decision on the size of the data cells was required. On machines with 36 or 48 bit words, two words per data cell were sufficient. The XDS-9300 has a word size of 24 bits and requires 15 bits for maximum addressibility. It was decided therefore to allocate four machine words per data cell. This provides room for a 3 bit ID field, and two 21 bit pointer fields (LNKL and LNKR) in the first two words. The remaining 3 bit field is unused. The reason for the extra 6 bits per pointer is discussed below. The remaining two words comprise the DATUM field of the cell. Since two words are required on the 9300 for floating point representation, a two word DATUM had the additional benefit of allowing the storage of floating point data.

XDS FORTRAN IV provides the user with the option to intermix Fortran and assembly language (XDS SYMBOL) statements. This technique was used in writing the primitives. The coding of the

branching and parameter passing was left to the Fortran compiler and only those instructions necessarily on the assembly language level were so specified.

One problem peculiar to the 9300 was discovered in the routines written by Weizenbaum. Many function and variable names were implicitly of integer type. On the majority of machines this would cause no difficulty because assignment of values to those functions and variables is on the primitive level and hence no type incompatibilities or data conversions occur. The XDS-9300 allocates only one word of storage for integer type variables, therefore, for example, the NAME of a ring could not be returned by a function called LIST. The solution to this naming difficulty required that all function and variable names which were assigned the values of data items be changed to real valued type. Other than this modification little was required in the way of changes to the non-primitive routines.

Once the implmentation was complete and tested, ways to minimize the main memory requirements of resident SLIP routines were examined. As implemented on the XDS-9300, SLIP consisted of 102 subprograms (or approximately thirteen hundred Fortran and SYMBOL statements). By placing the SLIP subprograms in the system Primary Library only those required by the user's program will be core resident at execution time. In most instances a considerable saving of memory is achieved over the alternative of making all of the routines

resident at execution time. If the user's program is so constructed as to take advantage of the XDS Monitor's overlay package further economies in storage can be gained.

As originally implemented SLIP required the user to define the amount of core available to the SLIP structure. Cells were allocated from this space and the program terminated when the allocated space was exhausted. Only a relatively small percentage of 9300 core is required by the resident routines of the Monitor and the AGT-10 interface leaving the bulk of the 32K main memory available to the user. It would require a large user program and model to exhaust the remaining core, particularly in view of the excellent system overlay features.

Anticipating the future development of software packages to further support the use of the 9300 as a CAD system, it was decided that an overlay capability for the SLIP structure would be advantageous. This overlay (or paging) of the SLIP structure would offer a present benefit of allowing the building of very large models should the need arise.

SLIP was found to lend itself readily to paging. Since originally SLIP was restricted to a fixed sized structure space with an available storage list for that space, it was a logical extension to conceptualize that structure space as one of an arbitrary number of pages, each with its unique available storage list. It was desired to maintain the pages on the system drum in a random access file. Unfortunately the

the random access Monitor routines have only recently been made available by XDS Corporation and have not yet been received. The use of sequential files was the only alternative and so the pages were stored on the drum in that manner.

The use of sequential files for paging has some rather serious disadvantages. Two sequential files must be maintained because when a page swap occurs the page presently in core must be re-written into the file if it has been modified. Sequential files as a rule must be rewritten in their entirety, hence each time a page is paged out all the pages in the file must be copied back into the file. This requires the standard "Old Master-New Master" file scheme. In this method each time an update to the file is required, the pages are read from the "Old Master" and copied onto the "New Master" until the page to be updated is reached. The update is then made from core, and the "Old Master" advanced one record. The remainder of the "New Master" is written as before. In the next update operation the "New Master" becomes the "Old Master" and the process is repeated. This is obviously not the most efficient usage of a random access device.

Sequential file operations on the 9300 are device independent so that paging can be done alternatively on magnetic tape merely by changing a control card. In some circumstances it might be advantageous to have, in effect, permanent storage of the model rather than re-creating it each time it is required.

Paging was incorporated into SLIP by taking advantage of the ease in which major changes in SLIP may be accomplished by modification of the primitive routines. The LNKL and LNKR fields were increased to accommodate an additional high order 6 bits. These bits indicate the page number of the page which contains the data cell specified by the low order 15 bits. Each pointer then still references a unique address.

All retrievals of data items from cells are done by two primitive functions. These primitives were modified to incorporate a check for residency of the page containing the cell to be accessed. Should the page not be resident, page swapping is initiated. In a similar fashion, the primitives which store data items in cells were modified to call in the appropriate page as needed.

New pages are created by the non-primitive function which allocates data cells. Originally when the available storage list was emptied the program was terminated. In the paged form of SLIP when the current page becomes filled this function checks the other existing pages for available space. The search begins with the first page created and allocates the required cell from the first non-filled page. The search is simplified by the fact that the HEADER of the available storage list of each page is resident in core. Should all pages be filled a new page is careated and the cell allocation made.

SLIP's storage management mechanisms were maintained in the paged version with little modification. As cells of a page are allocated or freed the available storage list for that page is updated.

Initially SLIP in its paged form was found to function very inefficiently. Once structures were constructed which exceeded one page in size the overhead associated with even the most basic of SLIP operations became totally unacceptable. Analysis of the problem revealed the source of the inefficiency.

As rings are constructed each new cell is inserted as a rule either to the immediate right or left of the HEADER. The insertion operation then requires that either the LNKR or the LNKL of the HEADER be modified to point to the new cell. Therefore once the ring is carried over onto a new page, each insertion thereafter requires a paging operation to access the HEADER followed by a paging operation to make the insertion.

A similar situation arises when retrieving information from SLIP structures. As previously discussed, a READER cell is required to advance through the structure. If the READER cell was allocated on a page different from the one containing the ring which is to be "read" the following sequence occurs. The READER is accessed to obtain the pointer to the cell to be "read". The appropriate page is swapped in and the retrieval made. The page containing the READER is swapped back in and the pointer to the next cell to be "read" is updated.

It was apparent that the root of the problem was located in the requirement that a ring HEADER or a READER cell be accessed for virtually every SLIP operation. Should the ring extend over several pages or the READER be on a separate page from the ring it was to

access, then the number of paging operations required became prohibitively large. A very significant contributing factor was the high overhead associated with each paging operation, due to the sequential page files. Little could be done to reduce the overhead of the paging operation, so the paging scheme had to be improved.

If the HEADER and READER cells were permanently core resident, an order of magnitude reduction in the number of paging operations required to process a typical SLIP structure could be achieved. The desired residency of these cells was accomplished by the creation of page zero. Heretofore pages had been numbered starting with one and were all overlayable with only one page resident at a time. By flightly modifying existing routines, a page zero was defined which remained resident at all times. All HEADER and READER cells are allocated from this page. Should page zero become filled, further allocations will be made from the next available page and a warning given to the SLIP user that paging efficiency has been seriously degraded.

The page zero modification achieved a marked improvement in the operation of SLIP in a paged environment. Test programs exercising a general mix of SLIP functions demonstrated a reasonable degree of efficiency. As in most paging schemes, "worst case" situations will result in an excessive amount of computer time being devoted to I/O associated with page swapping. Two SLIP subprograms RLSTEQL and RLSSCPY are involved in the "worst cases" of the

paging scheme. RLSTEQL examines two ring structures for equality while RLSSCPY creates a copy of a ring structure. Should the page where the second structure resides in the case of RLSTEQL, or on which the copy is being constructed in the case of RLSSCOPY, be different from the page containing the first structure then excessive paging results. A paging operation is then required for each data cell of each structure. The programmer will probably have little call to use these two functions but if they should be required care should be taken to avoid, if possible, having the two structures on separate pages. Further reductions in the overhead of paging can undoubtedly be made by increasing the efficiency of the paging algorithms, but the greatest improvement will be achieved by converting the page files from sequential to random access. The conversion is discussed in the following section.

Reference 9 provides a reasonably detailed description of the important SLIP subprograms. Those subprograms which were modified to support the paging scheme and those added for that purpose are discussed below in order to complete the documentation.

INITAS, a non-primitive subroutine which established the public lists [Ref. 9] and originally initialized the structure space and the available storage list, was modified to create page zero. The initialization of the structure space and available storage list was delegated to a new subroutine named INITIAL. This routine is called each time a new page is created.

NUCELL, a non-primitive function which does all data cell allocation, was modified to initiage paging when the current page becomes filled. This function originally was called with a dummy argument to satisfy Fortran conventions. NUCELL is now called with an actual parameter which causes allocation from page zero if the cell required is to be used as a HEADER or a READER.

RCELL, a non-primitive function which returns freed cells to the available storage list, was modified so as to return the cell to the available storage list of the appropriate page.

NPAGECK is a new primitive function which is called with a pointer as an argument. This function examines the high order 6 bits of the pointer and compares the page number with the number of the current page. If they are not equal a page swap is initiated. NPAGECK returns the value of the low order 15 bits (i. e. the "local" address within the page).

PAGING is a new non-primitive subroutine which is called with a page number as an argument. If the page number is that of an existing page the subroutine OLDPAGE is called. Otherwise a call is made to the subroutine NEWPAGE.

OLDPAGE is a new non-primitive called with a page number as an argument which initiates an update of the page file (pages out the current page) and reads in the page specified.

NEWPAGE is a new non-primitive called with a page number as an argument which initiates an update of the page file and creates a new page in core.

UPDATE is a new non-primitive subroutine which performs the update operation on the page files. This routine initiates a copy of the "Old Master" onto the "New Master" until the current page is reached. At this point the current page is copied onto the "New Master" and the copy from the "Old Master" continued until the end of file.

COPY is a new non-primitive subprogram which performs the copy operation specified by UPDATE.

The paged implementation of SLIP no longer requires the user to specify the size of the structure space. The user is required to declare on the control card prior to the execution of his Fortran program the devices to which the files "Old Master" and "New Master" are to be assigned. These files have the logical device numbers of 1 and 2 respectively. The XDS REAL TIME MONITOR manual specifies how assignments of logical device codes to physical devices may be made. Additionally the user must include in his Fortran program a CALL INITAS statement (without arguments) prior to a call to any other SLIP subprograms. Satisfaction of the above two requirements provides the user with the full range of SLIP operations with little practical restriction on the size of the models he can construct.

IV. CONCLUSION

Several aspects of SLIP as implemented on the XDS-9300 are subject to possible modification. The size of pages are rather arbitrarily set at 250 data cells per page. How appropriate this page size will be as SLIP is used to construct the models for a wide range of problems is yet to be determined. A study is needed to determine the amount of storage required by the average model and how paging overhead varies with page size. The results of such a study would enable a page size to be established which provided the best operational efficiency for the average model. An alternative would be to modify SLIP so as to allow the user to specify the page size for his specific application.

Incorporation of the Monitor update which will provide the capability for random access files on the 9300 will greatly reduce paging overhead. The sequential page files of the current implementation require a READ and a WRITE for each page in the file when a page is paged out. Each page in the file is then read until the page to be paged in is reached. Conversion to a random access page file would require only a single WRITE for the page out operation and a single READ for the page in operation. Additionally the duplication of files would be eliminated.

The conversion of SLIP to accommodate a random access page file can be accomplished in a straight forward manner. The subroutines COPY and UPDATE will no longer be needed and can be discarded. Subroutines NEWPAGE and OLDPAGE will require modification. Modifications to NEWPAGE consist of removing the coding which accomplishes the sequential page file operations and replacing it with a single RAD Fortran WRITE statement (see the XDS FORTRAN IV manual) to write the current page in the file. In similar fashion OLDPAGE is modified to random access WRITE the current page and READ the page to be swapped in.

A major modification to SLIP which would increase the storage utilization efficiency of SLIP would be the construction of varying length data cells. These cells which could contain an arbitrary number of data items would be useful for the storage of blocks of related data. For example the co-ordinates of the points required for the display of an electronic component could be displayed in a single data cell rather than as a ring of data cells each containing a single set of co-ordinates. The use of variable length data cells, however, greatly increases the complexity of the storage management scheme.

SLIP, as implemented, appears to provide a modeling capability of sufficient flexibility to cope with the problems expected to be run on the XDS-9300. No practical limitation is placed on the size of the models which can be constructed and adequate main memory is

available for large resident application programs. Efficiency of the sequential file paging scheme is reasonable except in a few situations previously described which can be avoided with foresight. A random file paging scheme should have good efficiency under all circumstances. The SLIP user is not required to learn a new language, and once the user gains familiarity with the functions of the SLIP routines he can easily incorporate the SLIP structure into existing programs.

The process by which a specific data structure was selected and implemented at a particular installation has been described. The factors behind the decisions made during the course of that project were highly machine and user dependent. A similar project at another installation might easily result in the implementation of a modeling scheme greatly different from the one implemented at N.P.S. Regardless of installation, the selection of a data structure for any medium sized computer must involve a determination of what capabilities are expected of the structure selected, as well as an analysis of the structures available and the requirements each places on user and machine. Finally a compromise must be made among structure capability, ease of use and machine resources required.

```
      FUNCTION ADVLEL(LR,N)
X     PRINT 999
X 999 FORMAT(8H ADVLEL )
      N=ADVLL(LR,0,0)
      IF (N)1,2,1
    2 ADVLEL=REED(LR)
    1 RETURN
      END


      FUNCTION ADVLER(LR,N)
X     PRINT 999
X 999 FORMAT(8H ADVLER )
      N=ADVLR(LR,0,0)
      IF (N)1,2,1
    2 ADVLER=REED(LR)
    1 RETURN
      END


      FUNCTION ADVLL(LR,J,K)
X     PRINT 999
X 999 FORMAT(8H ADVLL  )
      CLR=CONT(LR)
    5 LK=LNKL(CONT(LNKL(CLR)))
      CAND=CONT(LK)
      CALL SETDIR(-1,LK,-1,CLR)
      IF (ID(CAND)-2)1,2,1
    1 IF (ID(CAND)-J)3,4,3
    3 IF (ID(CAND)-K)5,4,5
    4 ADVLL=0.0
      GO TO 6
    2 ADVLL=-1.0
    6 CALL STRIND(CLR,LR)
      RETURN
```

```
      END


      FUNCTION ADVLNL(LR,N)
X     PRINT 999
X 999 FORMAT(8H ADVLNL )
      N=ADVLL(LR,1,1)
      IF (N)1,2,1
    2 ADVLNL=REED(LR)
    1 RETURN
      END


      FUNCTION ADVLNR(LR,N)
X     PRINT 999
X 999 FORMAT(8H ADVLNR )
      N=ADVLR(LR,1,1)
      IF (N)1,2,1
    2 ADVLNR=REED(LR)
    1 RETURN
      END


      FUNCTION ADVLR(LR,J,K)
X     PRINT 999
X 999 FORMAT(8H ADVLR  )
      CLR=CONT(LR)
    5 LK=LNKR(CONT(LNKL(CLR)))
      CAND=CONT(LK)
      CALL SETDIR(-1,LK,-1,CLR)
      IF (ID(CAND)-2)1,2,1
    1 IF (ID(CAND)-J)3,4,3
    3 IF (ID(CAND)-K)5,4,5
    4 ADVLR=0.0
      GO TO 6
```

```
    2 ADVLR=-1.0
    6 CALL STRIND(CLR,LR)
      RETURN
      END


      FUNCTION ADVLWL(LR,N)
X     PRINT 999
X 999 FORMAT(8H ADVLWL )
      N=ADVLL(LR,1,0)
      IF (N)1,2,1
    2 ADVLWL=REED(LR)
    1 RETURN
      END


      FUNCTION ADVLWR(LR,I)
X     PRINT 999
X 999 FORMAT(8H ADVLWR )
      I=ADVLR(LR,1,0)
      IF (I)1,2,1
    2 ADVLWR=REED(LR)
    1 RETURN
      END


      FUNCTION ADVSEL(LR,N)
X     PRINT 999
X 999 FORMAT(8H ADVSEL )
      N=ADVSL(LR,0,0)
      IF (N)1,2,1
    2 ADVSEL=REED(LR)
    1 RETURN
      END
```

```
      FUNCTION ADVSER(LR,N)
X     PRINT 999
X 999 FORMAT(8H ADVSER )
      N=ADVSR(LR,0,0)
      IF (N)1,2,1
    2 ADVSER=REED(LR)
    1 RETURN
      END


      FUNCTION ADVSL(L,J,K)
X     PRINT 999
X 999 FORMAT(8H ADVSL  )
      Z=1.
      R=CONT(L)
      CAND=CONT(LNKL(R))
      IF (ID(CAND)-1)1,6,1
    1 LCP=LNKL(CAND)
      CALL SETDIR(-1,LCP,-1,R)
      CAND=CONT(LCP)
      IF (ID(CAND)-2)3,4,3
    3 IF (ID(CAND)-J)7,8,7
    7 IF (ID(CAND)-K)5,8,5
    5 IF (ID(CAND)-1)1,6,1
    6 M=NUCELL(Z)
      CALL STRIND(R,M)
      CALL STRIND(CONT(L+2),M+2)
      CALL SETIND(-1,INHALT(LCP+2),LCNTR(L)+1,L+2)
      CALL SETDIR(-1,-1,M,R)
      CAND=CONT(INHALT(LNKL(R)+2))
      GO TO 1
    4 IF (LCNTR(L))9,10,9
   10 ADVSL=-1.0
      GO TO 12
    9 LK=LNKR(R)
```

```
      R=CONT(LK)
      CALL STRIND(CONT(LK+2),L+2)
      CAND=CONT(LNKL(R))
      CALL RCELL(LK)
      GO TO 1
    8 ADVSL=0.0
   12 CALL STRIND(R,L)
      RETURN
      END


      FUNCTION ADVSNL(LR,N)
X     PRINT 999
X 999 FORMAT(8H ADVSNL )
      N=ADVSL(LR,1,1)
      IF (N)1,2,1
    2 ADVSNL=REED(LR)
    1 RETURN
      END


      FUNCTION ADVSNR(LR,N)
X     PRINT 999
X 999 FORMAT(8H ADVSNR )
      N=ADVSR(LR,1,1)
      IF (N)1,2,1
    2 ADVSNR=REED(LR)
    1 RETURN
      END


      FUNCTION ADVSR(L,J,K)
X     PRINT 999
X 999 FORMAT(8H ADVSR  )
      Z=1.
```

```
      R=CONT(L)
      CAND=CONT(LNKL(R))
      IF (ID(CAND)-1)1,6,1
    1 LCP=LNKR(CAND)
      CALL SETDIR(-1,LCP,-1,R)
      CAND=CONT(LCP)
      IF (ID(CAND)-2)3,4,3
    3 IF (ID(CAND)-J)7,8,7
    7 IF (ID(CAND)-K)5,8,5
    5 IF (ID(CAND)-1)1,6,1
    6 M=NUCELL(Z)
      CALL STRIND(R,M)
      CALL STRIND(CONT(L+2),M+2)
      CALL SETIND(-1,INHALT(LCP+2),LCNTR(L)+1,L+2)
      CALL SETDIR(-1,-1,M,R)
      CAND=CONT(INHALT(LNKL(R)+2))
      GO TO 1
    4 IF (LCNTR(L))9,10,9
   10 ADVSR=-1.0
      GO TO 12
    9 LK=LNKR(R)
      R=CONT(LK)
      CALL STRIND(CONT(LK+2),L+2)
      CAND=CONT(LNKL(R))
      CALL RCELL(LK)
      GO TO 1
    8 ADVSR=0.0
   12 CALL STRIND(R,L)
      RETURN
      END


      FUNCTION ADVSWL(LR,N)
X     PRINT 999
X 999 FORMAT(8H ADVSWL )
```

```
      N=ADVSL(LR,1,0)
      IF (N)1,2,1
    2 ADVSWL=REED(LR)
    1 RETURN
      END

      FUNCTION ADVSWR(LR,N)
X     PRINT 999
X 999 FORMAT(8H ADVSWR )
      N=ADVSR(LR,1,0)
      IF (N)1,2,1
    2 ADVSWR=REED(LR)
    1 RETURN
      END

      FUNCTION AITSVAL(AT,LST)
X     PRINT 999
X 999 FORMAT(8H AITSVAL)
      IF (LNKL(CONT(LST+2)))3,4,3
    3 M=MADATR(AT,LST)
      IF (M+1)1,2,1
    1 AITSVAL=CONT(LNKR(CONT(M))+2)
      RETURN
    4 CALL DERROR(LST)
    2 AITSVAL=0.
      RETURN
      END

      FUNCTION ANEWVAL(AT,VAL,LST)
X     PRINT 999
X 999 FORMAT(8H ANEWVAL)
      M=MADATR(AT,LST)
```

```
      IF (M+1)2,1,2
    2 ANEWVAL=SUBST(VAL,LNKR(CONT(M)))
      RETURN
    1 CALL RLDATVL(AT,VAL,LST)
      ANEWVAL=0.
      RETURN
      END


      FUNCTION ANOATVL(AT,LST)
X     PRINT 999
X 999 FORMAT(8H ANOATVL)
      M=MADATR(AT,LST)
      IF (M+1)2,1,2
    2 ANOATVL=DELETE(LNKR(CONT(M)))
      CALL DELETE(M)
      RETURN
    1 ANOATVL=0.
      RETURN
      END


      FUNCTION BOT(P)
X     PRINT 999
X 999 FORMAT(8H BOT     )
      BOT=CONT(LNKL(CONT(LOCT(P)))+2)
      RETURN
      END


      FUNCTION CLIST(X)
X     PRINT 999
X 999 FORMAT(8H CLIST   )
      Z=1.
      M=NUCELL(Z)
```

```
      CALL SETDIR(0,M,M,CLIST)
      CALL SETIND(2,M,M,M)
      IF (X-9)2,1,2
    2 CALL SETIND(-1,-1,1,M+2)
      X=CLIST
    1 RETURN
      END


      FUNCTION CONT(I)
X     PRINT 999
X 999 FORMAT(8H CONT   )
      M=NPAGECK(I)
S     LDX M,1
S     LDD 0,1
S     STD CONT
      RETURN
      END


      SUBROUTINE COPY(R1,R2)
      GLOBAL CELLS(500),MNEW,MOLD,CORE(500)
X     PRINT 999
X 999 FORMAT(8H COPY   )
      DO 3 I=R1,R2
      CALL BUFFERIN(MOLD,1,CORE,1000,K)
      CALL BUFFEROUT(MNEW,1,CORE,1000,K)
    3 CONTINUE
      RETURN
      END


      FUNCTION DELETE(K)
X     PRINT 999
X 999 FORMAT(8H DELETE )
```

```
      IF (ID(CONT(K))-2)1,2,1
    2 PRINT 901
      PRINT 903, CONT(K)
      PRINT 903, CONT(K+2)
  903 FORMAT(1H ,016)
      DELETE=C.0
      RETURN
  901 FORMAT (1H1,97HAN ATTEMPT HAS BEEN MADE TO DELETE A HEADER - ZERO
     1HAS BEEN DELIVERED AND THE PROGRAM CONTINUED. )
    1 DELETE=CONT(K+2)
      LL=LNKL(CONT(K))
      LR=LNKR(CONT(K))
      CALL RCELL(K)
      CALL SETIND(-1,-1,LR,LL)
      CALL SETIND(-1,LL,-1,LR)
      RETURN
      END


      SUBROUTINE DERROR(LST)
X     PRINT 999
X 999 FORMAT(8H DERROR )
      PRINT 900,LST
      PRINT 901
      RETURN
  900 FORMAT (1H1,20X,016)
  901 FORMAT (20X,44HATTRIBURE-VALUE LIST REQUIRED BUT NOT FOUND )
      END


      FUNCTION EQUAL(X,Y)
X     PRINT 999
X 999 FORMAT(8H EQUAL  )
      IF (X.EQ.Y) GO TO 1
      EQUAL=-1
```

```
      GO TO 2
    1 EQUAL=0
    2 RETURN
      END


      FUNCTION ID(I)
X     PRINT 999
X 999 FORMAT(8H ID     )
      M=1
S     LDA *I
S     SHIFT 021025
S     STA ID
      RETURN
      END


      SUBROUTINE INITAS
      COMMON AVSL(0:100),X(100)
      GLOBAL ZPAGE(200),PAGE,PAGEMAX,MNEW,MASTER
      DIMENSION W(200)
X     PRINT 999
X 999 FORMAT(8H INITAS )
      MASTER=1
      MNEW=1
      PAGE=1
      PAGEMAX=0
      AVSL(0)=0.
      DO 1 I=1,100
      AVSL(I)=0.
      J=2*I-1
      CALL STRDIR(STRDIR(STRDIR(0.0,X(I)),W(J)),W(J+1))
      CALL SETDIR(0,MADDV(W(J)),MADDV(W(J)),X(I))
      CALL SETDIR(2,MADDV(W(J)),MADDV(W(J)),W(J))
    1 CALL SETDIR(-1,-1,4095,W(J+1))
```

```
      DO 2 I=1,200
    2 ZPAGE(I)=0.
      DO 3 I=1,197,2
      CALL SETDIR(-1,-1,MADDV(ZPAGE(I+2)),ZPAGE(I))
    3 CONTINUE
      CALL SETDIR(0,MADDV(ZPAGE(199)),MADDV(ZPAGE(1)),AVSL(0))
      CALL INITIAL
      RETURN
      END


      SUBROUTINE INITIAL
      COMMON AVSL(0:100)
      GLOBAL CELLS(500),PAGE
X     PRINT 999
X 999 FORMAT(8H INITIAL)
      DO 1 I=1,500
    1 CELLS(I)=0.
      DO 2 I=1,498,2
      CALL SETDIR(-1,-1,MADDV(CELLS(I+2)),CELLS(I))
    2 CALL SETPAGE(1,1,CELLS(I))
      CALL SETDIR(0,MADDV(CELLS(499)),MADDV(CELLS(1)),AVSL(PAGE))
      CALL SETPAGE(1,1,AVSL(PAGE))
      CALL SETPAGE(1,1,CELLS(499))
      RETURN
      END


      FUNCTION INITRD(K)
X     PRINT 999
X 999 FORMAT(8H INITRD )
      CALL SETIND(-1,LNKL(K+2),-1,K)
      INITRD=K
      RETURN
      END
```

```
      FUNCTION INLSTL(M,N)
X     PRINT 999
X 999 FORMAT(8H INLSTL )
      L=LOCT(M)
      ITOP=LNKR(CONT(L))
      IBOT=LNKL(CONT(L))
      INLSTL=L
      CALL SETIND(-1,L,L,L)
      IPRE=LNKR(CONT(N))
      CALL SETIND(-1,IBOT,-1,N)
      CALL SETIND(-1,-1,ITOP,IPRE)
      CALL SETIND(-1,IPRE,-1,ITOP)
      CALL SETIND(-1,-1,N,IBOT)
      RETURN
      END


      FUNCTION INLSTR(M,N)
X     PRINT 999
X 999 FORMAT(8H INLSTR )
      L=LOCT(M)
      ITOP=LNKR(CONT(L))
      IBOT=LNKL(CONT(L))
      INLSTR=L
      CALL SETIND(-1,L,L,L)
      ISUC=LNKR(CONT(N))
      CALL SETIND(-1,-1,ITOP,N)
      CALL SETIND(-1,IBOT,-1,ISUC)
      CALL SETIND(-1,N,-1,ITOP)
      CALL SETIND(-1,-1,ISUC,IBOT)
      RETURN
      END


      FUNCTION IRALST(P)
```

```
X       PRINT 999
X 999   FORMAT(8H IRALST )
        Z=1.
        L=LOCT(P)
        CALL SETIND(-1,-1,LCNTR(L)-1,L+2)
        IRALST=LCNTR(L)
        IF (IRALST)1,2,1
    2   CALL MTLIST(P)
        N=LNKL(CONT(L+2))
        IF (N)3,4,3
    3   NEW=NUCELL(Z)
        CALL SETIND(1,-1,-1,NEW)
        CALL SETIND(-1,N,N,NEW+2)
        CALL RCELL(NEW)
    4   CALL RCELL(L)
    1   RETURN
        END


        FUNCTION IRARDR(K)
X       PRINT 999
X 999   FORMAT(8H IRARDR )
        IRARDR=LCNTR(K)
        M=K
    3   N=LNKR(CONT(M))
        CALL RCELL(M)
        IF (N)1,2,1
    1   M=N
        GO TO 3
    2   RETURN
        END


        FUNCTION LCNTR(K)
X       PRINT 999
```

```
X 999 FORMAT(8H LCNTR  )
      LCNTR=LNKR(CONT(K+2))
      RETURN
      END


      FUNCTION LISTMT(P)
X     PRINT 999
X 999 FORMAT(8H LISTMT )
      L=LOCT(P)
      IF (EQUAL(CONT(L),CONT(LNKR(CONT(L))))3,4,3
    4 LISTMT=0
      RETURN
    3 LISTMT=-1
      RETURN
      END


      FUNCTION LNKL(I)
X     PRINT 999
X 999 FORMAT(8H LNKL   )
      M=1
S     LDA =07777777
S     ETR *I
S     STA LNKL
      RETURN
      END


      FUNCTION LNKR(I)
X     PRINT 999
X 999 FORMAT(8H LNKR   )
      M=1
S     MPO I
S     LDA =07777777
```

```
S       ETR *I
S       STA LNKR
        RETURN
        END


        FUNCTION L3CT(K)
X       PRINT 999
X 999 F0RMAT(8H L0CT   )
        IF (NAMTST(K))1,2,1
    2 L0CT=K
        RETURN
    1 PRINT 901
        STOP
  901 FORMAT (1H1,94HA LIST WAS REQUIRED AS AN 0PERAND BUT WAS N0T F0UND
     1- THE PR0GRAM WAS REGRETFULLY TERMINATED . )
        END


        FUNCTION LPNTR(K)
X       PRINT 999
X 999 F0RMAT(8H LPNTR  )
        LPNTR=LNKL(C0NT(K))
        RETURN
        END


        FUNCTION LP0INT(L)
X       PRINT 999
X 999 F0RMAT(8H LP0INT )
        M=1
S       LDA =07777777
S       ETR *L
S       SHIFT 025011
S       SHIFT 021011
```

```
6       STA M
        LPRINT=M
        RETURN
        END


        FUNCTION LPURGE(LST)
X       PRINT 999
X 999 FORMAT(8H LPURGE )
        K=LRDRCV(LST)
        LPURGE=0
      3 X=ADVSWR(K,J)
      6 IF (J)1,2,1
      1 IF (NAMTST(X))3,4,3
      4 IF (LSTPRO(X,K))3,5,3
      5 L=LPNTR(K)
        X=ADVLWR(K,J)
        CALL DELETE(L)
        LPURGE=LPURGE+1
        GO TO 6
      2 CALL IRARDR(K)
        RETURN
        END


        FUNCTION LRDRCP(K)
X       PRINT 999
X 999 FORMAT(8H LRDRCP )
        Z=1.
        LRDRCP=NUCELL(Z)
        NEWR=LRDRCP
        NOW=K
      3 CALL STRIND(CONT(NOW),NEWR)
        CALL STRIND(CONT(NOW+2),NEWR+2)
        NOW=LNKR(CONT(NOW))
```

```
      IF (NEW)1,2,1
    1 NEW=NUCELL(Z)
      CALL SETIND(-1,-1,NEW,NEWR)
      NEWR=NEW
      GO TO 3
    2 RETURN
      END


      FUNCTION LRDROV(P)
X     PRINT 999
X 999 FORMAT(8H LRDROV )
      Z=1.
      LRDROV=NUCELL(Z)
      CALL SETIND(3,LOCT(P),0,LRDROV)
      CALL SETIND(0,P,0,LRDROV+2)
      RETURN
      END


      FUNCTION LSTPRO(L,K)
X     PRINT 999
X 999 FORMAT(8H LSTPRO )
      NEXT=K
    3 IF (LNKL(NEXT+2)-LNKR(L))1,2,1
    1 NEXT=LNKL(NEXT)
      IF (NEXT)3,4,3
    2 LSTPRO=0
      RETURN
    4 LSTPRO=-1
      RETURN
      END


      FUNCTION LVLRV1(K)
```

```
X      PRINT 999
X 999 FORMAT(8H LVLRV1 )
      LVLRV1=K
      IF (CONT(LVLRV1+2))2,3,2
    3 RETURN
    2 L=LNKR(CONT(LVLRV1))
      CALL STRIND(CONT(L),LVLRV1)
      CALL STRIND(CONT(L+2),LVLRV1+2)
      CALL RCELL(L)
      RETURN
      END


      FUNCTION LVLRVT(K)
X      PRINT 999
X 999 FORMAT(8H LVLRVT )
      LVLRVT=K
    1 IF (CONT(LVLRVT+2))2,3,2
    3 RETURN
    2 L=LNKR(CONT(LVLRVT))
      CALL STRIND(CONT(L),LVLRVT)
      CALL STRIND(CONT(L+2),LVLRVT+2)
      CALL RCELL(L)
      GO TO 1
      END


      FUNCTION MADATR(AT,LST)
X      PRINT 999
X 999 FORMAT(8H MADATR )
      LSTDES=LNKL(CONT(LST+2))
      IF (LSTDES)1,4,1
    1 MADATR=LNKR(CONT(LSTDES))
    8 IF (ID(CONT(MADATR))-2)3,4,3
    3 IF (EQUAL(CONT(MADATR+2),AT))5,6,5
```

```
    5 M=LNKR(CONT(MADATR))
      IF (ID(CONT(M))-2)7,4,7
    7 MADATR=LNKR(CONT(M))
      GO TO 5
    4 MADATR=-1
    6 RETURN
      END


      FUNCTION MTLIST(P)
      COMMON AVSL(0:100)
      GLOBAL PAGE
X     PRINT 999
X 999 FORMAT(8H MTLIST )
      M=LOCT(P)
      IF (LISTMT(P))3,4,3
    3 LR=LNKR(CONT(M))
      LL=LNKL(CONT(M))
      CALL SETIND(-1,M,M,M)
      IF (LNKL(M).EQ.LPOINT(M)) GO TO 5
      CALL SETIND(-1,-1,LR,LNKL(AVSL(PAGE)))
      CALL SETDIR(-1,LL,-1,AVSL(PAGE))
      CALL SETIND(-1,-1,0,LNKL(AVSL(PAGE)))
    4 MTLIST=M
      RETURN
    5 CALL SETIND(-1,-1,LR,LNKL(AVSL(0)))
      CALL SETDIR(-1,LL,-1,AVSL(0))
      CALL SETIND(-1,-1,0,LNKL(AVSL(0)))
      GO TO 4
      END


      FUNCTION MADOV(I)
X     PRINT 999
X 999 FORMAT(8H MADOV  )
```

```
      M=1
S     LDA =077777
S     ETR I
S     STA MADBV
      RETURN
      END


      FUNCTION NAMTST(K)
X     PRINT 999
X 999 FORMAT(8H NAMTST )
      IF (LNKL(K)-LNKR(K))1,4,1
    4 IF (ID(CONT(K))-2)1,2,1
    2 IF (CONT(LNKR(CONT(LNKL(CONT(K)))))-CONT(K))1,3,1
    3 NAMTST=0
      RETURN
    1 NAMTST=-1
      RETURN
      END


      FUNCTION NEWBOT(P,Q)
X     PRINT 999
X 999 FORMAT(8H NEWBOT )
      NEWBOT=NXTLFT(P,LOCT(Q))
      RETURN
      END


      SUBROUTINE NEWPAGE(PAGENEW)
      GLOBAL CELLS(500),PAGE,PAGEMAX,MNEW,CORE(500)
X     PRINT 999
X 999 FORMAT(8H NEWPAGE)
      IF (PAGE.GT.PAGEMAX) GO TO 1
      CALL UPDATE
```

```
      GO TO 4
    1 DO 2 I=1,PAGEMAX
      CALL BUFFERIN(MNEW,1,CORE,1000,K)
    2 CONTINUE
      CALL BUFFEROUT(MNEW,1,CELLS,1000,K)
      REWIND MNEW
      PAGEMAX=PAGEMAX+1
    4 PAGE=PAGENEW
      CALL INITIAL
      RETURN
      END


      FUNCTION NEWTOP(P,Q)
X     PRINT 999
X 999 FORMAT(8H NEWTOP )
      NEWTOP=NXTRGT(P,LOCT(Q))
      RETURN
      END


      FUNCTION NPAGECK(L)
      GLOBAL PAGE
X     PRINT 999
X 999 FORMAT(8H NPAGECK)
      N=1
S     LDA =007700000
S     ETR *L
S     SHIFT 021017
S     STA N
S     LDA =077777
S     ETR *L
S     STA NPAGECK
      IF (N.EQ.PAGE) GO TO 1
      IF (N.EQ.0) GO TO 1
```

```
      P=N
      CALL PAGING(P)
    1 RETURN
      END


      FUNCTION NUCELL(X)
      COMMON AVSL(0:100)
      GLOBAL PAGE,PAGEMAX
X     PRINT 999
X 999 FORMAT(8H NUCELL )
      IF (X.EQ.1.) GO TO 5
    6 M=LNKR(AVSL(PAGE))
      IF (LPOINT(M)) 1,2,1
    2 DO 11 P=1.,PAGEMAX
      M=LNKR(AVSL(P))
      IF (LPOINT(M)) 12,11,12
   11 CONTINUE
      CALL PAGING(P+1.)
      M=LNKR(AVSL(PAGE))
      GO TO 1
   12 CALL PAGING(P)
    1 IF (ID(CONT(M))-1)3,4,3
    4 CALL IRALST(CONT(M+2))
    3 CALL SETDIR(-1,-1,LNKR(CONT(M)),AVSL(PAGE))
   10 CALL STRIND(0.,M)
      CALL STRIND(0.0,M+2)
      NUCELL=M
      RETURN
    5 M=LNKR(AVSL(0))
      IF (M)7,13,7
   13 PRINT 201
      GO TO 6
    7 IF (ID(CONT(M))-1) 8,9,8
    9 CALL IRALST(CONT(M+2))
```

```
    8 CALL SETDIR(-1,-1,LNKR(CONT(M)),AVSL(0))
      GO TO 10
  901 FORMAT(1H ,56HPAGE ZERO FILLED-PAGING EFFICIENCY IS SERIOUSLY DEGR
     1ADED)
      END


      FUNCTION NXTLFT(M,A)
X     PRINT 999
X 999 FORMAT(8H NXTLFT )
      Z=0.
      IL=NJCELL(Z)
      NXTLFT=IL
      LL=LNKL(CONT(A))
      CALL SETIND(-1,-1,IL,LL)
      CALL SETIND(-1,IL,-1,A)
      CALL SETIND(0,LL,A,IL)
      IF (NAMTST(M))1,2,1
    2 CALL SETIND(1,-1,-1,IL)
      CALL SETIND(-1,-1,LCNTR(M)+1,M+2)
    1 CALL STRIND(M,IL+2)
      RETURN
      END


      FUNCTION NXTRGT(M,A)
X     PRINT 999
X 999 FORMAT(8H NXTRGT )
      Z=0.
      IR=NJCELL(Z)
      LR=LNKR(CONT(A))
      CALL SETIND(-1,IR,-1,LR)
      CALL SETIND(-1,-1,IR,A)
      CALL SETIND(0,A,LR,IR)
      IF (NAMTST(M))1,2,1
```

```
    2 CALL SETIND(1,-1,-1,IR)
      CALL SETIND(-1,-1,LCNTR(M)+1,M+2)
    1 CALL STRIND(M,IR+2)
      RETURN
      END


      SUBROUTINE OLDPAGE(PAGENEW)
      GLOBAL CELLS(500),PAGE,PAGEMAX,MNEW
X     PRINT 999
X 999 FORMAT(8H OLDPAGE)
      CALL UPDATE
      DO 1 I=1,PAGENEW
      CALL BUFFERIN(MNEW,1,CELLS,1000,K)
    1 CONTINUE
      REWIND MNEW
      PAGE=PAGENEW
      RETURN
      END


      SUBROUTINE PAGING(PAGENEW)
      GLOBAL PAGE,PAGEMAX
X     PRINT 999
X 999 FORMAT(8H PAGING )
      IF (PAGENEW.GT.PAGEMAX) GO TO 1
      CALL OLDPAGE(PAGENEW)
      GO TO 2
    1 CALL NEWPAGE(PAGENEW)
    2 RETURN
      END


      FUNCTION PARMT2(A,B)
      COMMON AVSL(0:100),W(100)
```

```
X       PRINT 999
X 999 F0RMAT(8H PARMT2 )
      CALL NEWT0P(A,W(1))
      CALL NEWT0P(B,W(2))
      PARMT2=A
      RETURN
      END

      FUNCTI0N P0PB0T(P)
X       PRINT 999
X 999 F0RMAT(8H P0PB0T )
      P0PB0T=DELETE(LNKL(C0NT(L0CT(P))))
      RETURN
      END

      FUNCTI0N P0PT0P(P)
X       PRINT 999
X 999 F0RMAT(8H P0PT0P )
      P0PT0P=DELETE(LNKR(C0NT(L0CT(P))))
      RETURN
      END

      SUBR0UTINE PRESRV(N)
      C0MM0N AVSL(0:100),W(100)
X       PRINT 999
X 999 F0RMAT(8H PRESRV )
      D0 1 I=1,N
    1 CALL NEWT0P(T0P(W(I)),W(I))
      RETURN
      END
```

```
      SUBROUTINE PRLSTS(OUTLST,I)
      EQUIVALENCE (KOUT,OUT)
X     PRINT 999
X 999 FORMAT(8H PRLSTS )
  900 FORMAT (1H1,20X,10HBEGIN LIST)
  901 FORMAT (21X,8HEND LIST)
  902 FORMAT (21X,I14)
  903 FORMAT (21X,13HBEGIN SUBLIST)
  904 FORMAT (21X,13HEND SUBLIST  )
  905 FORMAT (21X,A8)
  906 FORMAT (21X,F10.4)
  907 FORMAT (21X,13HEMPTY SUBLIST)
      PRINT 900
      LR=LRDROV(OUTLST)
      LEVEL=0
    7 X=ADVSWR(LR,K)
      IF (K)1,2,1
    2 IF (LEVEL-LCNTR(LR))21,22,23
   22 IF (NAMTST(X))3,4,3
    4 IF (LISTMT(X))5,6,5
    6 PRINT 907
      GO TO 7
    5 PRINT 903
      LEVEL=LEVEL+1
      GO TO 7
    3 GO TO (11,12,13)I
   11 OUT=X
      PRINT 902,KOUT
      GO TO 7
   12 OUT=X
      PRINT 905,KOUT
      GO TO 7
   13 PRINT 906,X
      GO TO 7
   23 PRINT 904
```

```
      LEVEL=LEVEL-1
      GO TO 2
    1 IF (LEVEL-LCNTR(LR))21,32,33
   33 PRINT 904
      LEVEL=LEVEL-1
      GO TO 1
   32 PRINT 901
      CALL RCELL(LR)
   21 RETURN
      END


      SUBROUTINE RCELL(CELL)
      COMMON AVSL(0:100)
X     PRINT 999
X 999 FORMAT(8H RCELL  )
      IF (LNKL(CELL).EQ.LPOINT(CELL)) GO TO 2
      CALL NPAGECK(CELL)
    1 CALL SETIND(-1,-1,CELL,LNKL(AVSL(PAGE)))
      CALL SETDIR(-1,CELL,-1,AVSL(PAGE))
    3 CALL SETIND(-1,-1,0,CELL)
      RETURN
    2 CALL SETIND(-1,-1,CELL,LNKL(AVSL(0)))
      CALL SETDIR(-1,CELL,-1,AVSL(0))
      GO TO 3
      END


      FUNCTION REED(K)
X     PRINT 999
X 999 FORMAT(8H REED   )
      REED=CONT(LNKL(CONT(K))+2)
      RETURN
      END
```

```
      FUNCTION RESTOR(N)
      COMMON AVSL(0:100),W(100)
X     PRINT 999
X 999 FORMAT(8H RESTOR )
      RESTOR=N
      DO 1 I=1,N
    1 CALL POPTOP(W(I))
      RETURN
      END


      FUNCTION RLDATVL(AT,VL,LST)
X     PRINT 999
X 999 FORMAT(8H RLDATVL)
      IF (LNKL(CONT(LST+2)))1,2,1
    2 RLDATVL=RLISTAV(LST)
    1 CALL NXTRGT(VL,NXTLFT(AT,LNKL(CONT(LST+2))))
      RETURN
      END


      FUNCTION RLISTAV(LST)
X     PRINT 999
X 999 FORMAT(8H RLISTAV)
      RLISTAV=CLIST(0.)
      CALL SETIND(-1,LNKR(RLISTAV),-1,LST+2)
      RETURN
      END


      FUNCTION RLOFRDR(K)
X     PRINT 999
X 999 FORMAT(8H RLOFRDR)
      L=LNKL(CONT(K+2))
      CALL SETDIR(0,L,L,RLOFRDR)
```

```
      RETURN
      END


      FUNCTION RLSSCPY(LA)
      COMMON AVSL(0:100),W(100)
X     PRINT 999
X 999 FORMAT(8H RLSSCPY)
      ASSIGN 100 TO L9C0
      RLSSCPY=VISIT(L9C0,PARMT2(LRDR0V(LA),CLIST(9.)))
      RETURN
  100 RLC=T0P(W(2))
      RLR=T0P(W(1))
    5 X=ADVLWR(RLR,K)
      IF (K)1,2,1
    1 CALL RCELL(RLR)
      CALL TERM(RLC,REST0R(2))
    2 IF (NAMTST(X))3,4,3
    3 CALL NEWB0T(X,RLC)
      G0 T0 5
    4 CALL NEWB0T(VISIT(L9C9,PARMT2(LRDR0V(X),CLIST(9.))),T0P(W(2)))
      G0 T0 100
      END


      FUNCTION RLSTEQL(LA,LB)
      COMMON AVSL(0:100),W(100)
X     PRINT 999
X 999 FORMAT(8H RLSTEQL)
      ASSIGN 100 TO L9C9
      RLSTEQL=VISIT(L9C0,PARMT2(LRDR0V(LA),LRDR0V(LB)))
      RETURN
  100 RLA=T0P(W(1))
      RLB=T0P(W(2))
    8 XA=ADVLWR(RLA,KA)
```

```
      XB=ADVLWR(RLB,KB)
        IF (KA)1,2,1
    1   IF (KB)3,4,3
    2   IF (KB)4,6,4
    6   IF (EQUAL(XA,XB))7,8,7
    7   IF (NAMTST(XA))4,9,4
    9   IF (NAMTST(XB))4,10,4
   10   RLSTEQL=VISIT(LOCO,PARMT2(LRDROV(XA),LRDROV(XB)))
        IF (RLSTEQL)4,100,4
    3   CALL RCELL(RLA)
        CALL RCELL(RLB)
        CALL TERM(0.,RESTOR(2))
    4   CALL RCELL(RLA)
        CALL RCELL(RLB)
        CALL TERM(-1.,RESTOR(2))
        END


        FUNCTION RMADLFT(K)
X       PRINT 999
X 999   FORMAT(8H RMADLFT)
        RMADLFT=0.0
        M=LNKL(CONT(K))
        IF (ID(CONT(M))-2)3,2,3
S   3   LDA M
S       STA RMADLFT
        GO TO 1
    2   CALL SETDIR(0,M,M,RMADLFT)
    1   RETURN
        END


        FUNCTION RMADNBT(P,N)
X       PRINT 999
X 999   FORMAT(8H RMADNBT)
```

```
      L=LBCT(P)
      LL=0
      DO 1 I=1,N
    1 L=LNKL(CONT(L))
      IF (ID(CONT(L))-2)2,3,2
    3 CALL SETDIR(0,L,L,L)
S   2 LDP L
S     STD RMADNBT
      RETURN
      END


      FUNCTION RMADNTP(P,N)
X     PRINT 999
X 999 FORMAT(8H RMADNTP)
      L=LBCT(P)
      LL=0
      DO 1 I=1,N
    1 L=LNKR(CONT(L))
      IF (ID(CONT(L))-2)2,3,2
    3 CALL SETDIR(0,L,L,L)
S   2 LDP L
S     STD RMADNTP
      RETURN
      END


      FUNCTION RMADRGT(K)
X     PRINT 999
X 999 FORMAT(8H RMADRGT)
      RMADRGT=0.0
      M=LNKR(CONT(K))
      IF (ID(CONT(M))-2)3,2,3
S   3 LDA M
S     STA RMADRGT
```

```
      GO TO 1
    2 CALL SETDIR(0,M,M,RMADRGT)
    1 RETURN
      END


      FUNCTION RMAKEDL(L,X)
X     PRINT 999
X 999 FORMAT(8H RMAKEDL)
      CALL RMTDLST(X)
      RMAKEDL=X
      N=LOCT(X)
      K=LOCT(L)
      CALL SETIND(-1,K,-1,N+2)
      CALL SETIND(-1,-1,LCNTR(L)+1,K+2)
      RETURN
      END


      FUNCTION RMRKLSS(M,RLST)
X     PRINT 999
X 999 FORMAT(8H RMRKLSS)
      RMRKLSS=RLST
      LR=LRDROV(RMRKLST(M,RLST))
    3 X=ADVSNR(LR,K)
      IF (K)1,2,1
    2 CALL SETIND(M,-1,-1,LNKR(X)+2)
      GO TO 3
    1 CALL RCELL(LR)
      RETURN
      END


      FUNCTION RMRKLST(M,RLST)
X     PRINT 999
```

```
X 999 FORMAT(8H RMRKLST)
      RMRKLST=RLST
      CALL SETIND(M,-1,-1,LOCT(RLST)+2)
      RETURN
      END


      FUNCTION RMTDLST(RLST)
X     PRINT 999
X 999 FORMAT(8H RMTDLST)
      RMTDLST=RLST
      K=LNKL(CONT(LOCT(RLST)+2))
      KK=0
      IF (K)1,2,1
    1 CALL SETDIR(0,K,K,K)
      CALL MTLIST(K)
    2 RETURN
      END


      FUNCTION RNULSTL(LNKP,LNKH)
X     PRINT 999
X 999 FORMAT(8H RNULSTL)
      RNULSTL=CLIST(9.)
      IF (ID(CONT(LNKP))-2)1,2,1
    2 CALL SETIND(2,RNULSTL,RNULSTL,RNULSTL)
      RETURN
    1 LTOP=LNKR(CONT(LNKH))
      LSUC=LNKR(CONT(LNKP))
      CALL SETIND(-1,-1,LSUC,LNKH)
      CALL SETIND(-1,LNKH,-1,LSUC)
      CALL SETIND(2,LNKP,LTOP,RNULSTL)
      CALL SETIND(-1,-1,RNULSTL,LNKP)
      CALL SETIND(-1,RNULSTL,-1,LTOP)
      RETURN
```

```
      END


      FUNCTION RNULSTR(LNKP,LNKH)
X     PRINT 999
X 999 FORMAT(8H RNULSTR)
      RNULSTR=CLIST(9.)
      IF (ID(CONT(LNKP))-2)1,2,1
    2 CALL SETIND(2,RNULSTR,RNULSTR,RNULSTR)
      RETURN
    1 LBOT=LNKL(CONT(LNKH))
      LPRE=LNKL(CONT(LNKP))
      CALL SETIND(-1,LPRE,-1,LNKH)
      CALL SETIND(-1,-1,LNKH,LPRE)
      CALL SETIND(2,LBOT,LNKP,RNULSTR)
      CALL SETIND(-1,RNULSTR,-1,LNKP)
      CALL SETIND(-1,-1,RNULSTR,LBOT)
      RETURN
      END


      SUBROUTINE SETDIR(I,L,N,LOC)
X     PRINT 999
X 999 FORMAT(8H SETDIR )
      M=1
      K=1
      IF (I.EQ.-1) GO TO 10
S     LDA *I
S     SHIFT 025025
S     STA K
S     LDA =07777777
S     ETR *LOC
S     MRG K
S     STA *LOC
   10 IF (L.EQ.-1) GO TO  20
```

```
S       LDA =07777777
S       ETR *L
S       STA *L
S       LDA =070000000
S       ETR *L0C
S       MRG *L
S       STA *L0C
   20 IF (N.EQ.-1) G0 T0 30
S       LDA =07777777
S       ETR *N
S       STA K
S       LDA =070000000
S       MP0 L0C
S       ETR *L0C
S       MRG K
S       STA *L0C
   30 RETURN
       END


       SUBR0UTINE  SETIND(I,L,N,L0C)
X      PRINT 999
X 999 F0RMAT(8H SETIND )
       K=1
S      LDA =07777777
S      ETR *L0C
S      STA K
       K=NPAGECK(K)
S      LDX K,1
       IF (I.EQ.-1) G0 T0 10
S      LDA *I
S      SHIFT 025025
S      STA K
S      LDA =07777777
S      ETR 0,1
```

```
S       MRG K
S       STA 0,1
   10 IF (L.EQ.-1) GO TO 20
S       LDA =07777777
S       ETR *L
S       STA K
S       LDA =070000000
S       ETR 0,1
S       MRG K
S       STA 0,1
   20 IF (N.EQ.-1) GO TO 30
S       LDA =07777777
S       ETR *N
S       STA K
S       LDA =070000000
S       ETR 1,1
S       MRG K
S       STA 1,1
   30 RETURN
       END


       SUBROUTINE SETPAGE(I,J,ADDR)
       GLOBAL PAGE
X      PRINT 999
X 999 FORMAT(8H SETPAGE)
       M=PAGE
       JJ=J
S      LDA M
S      SHIFT 025017
S      STA M
S      LDA =007700000
S      ETR M
S      STA M
       IF (I.EQ.-1) GO TO 10
```

```
S    5 LDA =070077777
S      ETR *ADDR
S      MRG M
S      STA *ADDR
     10 IF (JJ.EQ.-1) GO TO 20
S      MPG ADDR
       JJ=-1
       GO TO  5
     20 RETURN
       END


       FUNCTION SEQLL(Z,N)
X      PRINT 999
X  999 FORMAT(8H SEQLL  )
       L=LNKL(Z)
       Z=CONT(L)
       SEQLL=CONT(L+2)
       N=ID(Z)-1
       RETURN
       END


       FUNCTION SEQLR(Z,N)
X      PRINT 999
X  999 FORMAT(8H SEQLR  )
       L=LNKR(Z)
       Z=CONT(L)
       SEQLR=CONT(L+2)
       N=ID(Z)-1
       RETURN
       END


       FUNCTION SEQRDR(LST)
```

```
X     PRINT 999
X 999 FORMAT(8H SEQRDR )
      SEQRDR=CONT(LOCT(LST))
      RETURN
      END


      FUNCTION SEQSL(Z,N)
X     PRINT 999
X 999 FORMAT(8H SEQSL  )
      IF (ID(Z)-1)4,5,4
    5 L=LNKL(CONT(CONT(LNKL(CONT(LNKR(Z)))+2)))
      GO TO 3
    4 L=LNKL(Z)
    3 IF (ID(CONT(L))-1)1,2,1
    1 SEQSL=CONT(L+2)
      Z=CONT(L)
      N=ID(Z)-1
      RETURN
    2 L=LNKL(CONT(CONT(L+2)))
      GO TO 3
      END


      FUNCTION SEQSR(Z,N)
X     PRINT 999
X 999 FORMAT(8H SEQSR  )
      IF (ID(Z)-1)4,5,4
    5 L=LNKR(CONT(CONT(LNKL(CONT(LNKR(Z)))+2)))
      GO TO 3
    4 L=LNKR(Z)
    3 IF (ID(CONT(L))-1)1,2,1
    1 SEQSR=CONT(L+2)
      Z=CONT(L)
      N=ID(Z)-1
```

```
      RETURN
    2 L=LNKR(CONT(CONT(L+2)))
      GO TO 3
      END


     FUNCTION STRDIR(DATA,CELL)
X     PRINT 999
X 999 FORMAT(8H STRDIR )
      STRDIR=DATA
S     LDP *DATA
S     STD *CELL
      RETURN
      END


      FUNCTION STRIND(DATA,CELL)
X     PRINT 999
X 999 FORMAT(8H STRIND )
      M=1
S     LDA *CELL
S     STA M
      M=NPAGECK(M)
S     LDX M,1
S     LDP *DATA
S     STD 0,1
      STRIND=DATA
      RETURN
      END


      FUNCTION SUBSBT(DAF,LST)
X     PRINT 999
X 999 FORMAT(8H SUBSBT )
      SUBSBT=SUBST(DAF,LNKL(CONT(LST)))
```

```
      RETURN
      END


      FUNCTION SUBST(D,N)
X     PRINT 999
X 999 FORMAT(8H SUBST  )
      LBACK=LNKL(CONT(N))
      SUBST=DELETE(N)
      CALL NXTRGT(D,LBACK)
      RETURN
      END


      FUNCTION SUBSTP(DAT,LST)
X     PRINT 999
X 999 FORMAT(8H SUBSTP )
      SUBSTP=SUBST(DAT,LNKR(CONT(LST)))
      RETURN
      END


      FUNCTION TERM(X,Y)
      COMMON AVSL(0:100),W(100)
X     PRINT 999
X 999 FORMAT(8H TERM   )
      K=0
      Z=POPTOP(W(100))
      TERM=X
S     LDA Z
S     STA K
S     LDP TERM
S     BRR K
      END
```

```
      FUNCTION TOP(P)
X     PRINT 999
X 999 FORMAT(8H TOP    )
      TOP=CONT(LNKR(CONT(LOCT(P)))+2)
      RETURN
      END


      SUBROUTINE UPDATE
      GLOBAL CELLS(500),PAGE,PAGEMAX,MNEW,MOLD,MASTER
Y     PRINT 999
X 999 FORMAT(8H UPDATE )
      IF (MASTER/2)1,2,1
    1 MNEW=1
      MOLD=2
      MASTER=1
      GO TO 3
    2 MNEW=2
      MOLD=1
      MASTER=2
    3 REWIND MNEW
      REWIND MOLD
      CALL COPY(1.,PAGE-1.)
      CALL BUFFEROUT(MNEW,1,CELLS,1000,K)
      IF (PAGE.LT.PAGEMAX) CALL BUFFERIN(MOLD,1,CELLS,1000,K)
      CALL COPY(PAGE+1.,PAGEMAX)
      IF (PAGE.GT.PAGEMAX) PAGEMAX=PAGEMAX+1
      REWIND MNEW
      REWIND MOLD
      RETURN
      END


      FUNCTION VISIT(LOC,X)
      COMMON AVSL(0:100),W(100)
```

```
X       PRINT 999
X 999   FORMAT(8H VISIT  )
        Z=0.0
S       LDA $-7
S       STA Z
        CALL NEWTOP(Z,W(100))
        RETURN LOC
        END
```

LIST OF REFERENCES

1. Knowlton, K. C., "A Programmer's Description of L^6," Communications of the ACM, v. 9, no. 8, p. 616-625, August 1966.

2. AED-O Programming Manual, Preliminary Release No. 2, MIT Electronic Systems Laboratory, November 1964.

3. Gray, J. C., "Compound Data Structure for Computer Aided Design: A Survey, "Proceedings of the 22nd National Conference of the ACM, 1967, p. 355-365.

4. Childs, D. L., "Description of A Set Theoretic Data Structure," AFIPS Conference Proceedings, Fall Joint Computer Conference, 1968, p. 557-564.

5. Rovner, P. D. and Feldman, J. A., "The Leap Language and Data Structure," Proceedings of IFIP Congress 1968, p. 579-585.

6. Sutherland, I. E., "Sketchpad: A Man-Machine Graphical Communication System," AFIPS Conference Proceedings, Spring Joint Computer Conference, 1963, p. 329-346.

7. Roberts, L. G., "Graphical Communications and Control Languages, "Second Congress on the Information System Sciences, p. 211-217, Spartan Books, 1964.

8. Lang, C. A. and Gray, J. C., "ASP-A Ring Implemented Associative Structure Package," Communications of the ACM, v. 11, no. 8, p. 550-555, August 1968.

9. Weizenbaum, J., "Symmetric List Processor, "Communications of the ACM, v. 6, no. 9, p. 524-536, September 1963.

10. The Rand Corporation Memorandum RM-6145-ARPA, A Survey of Data Structures for Interactive Graphics, by J. A. Hamilton, April 1970.

BIBLIOGRAPHY

Baugh, C. R., "Fodesar: FORTRAN Dependent Storage Allocation and Relocation Package," Proceedings of the 20th National Conference of the ACM, 1965, p. 315-324.

Bolles, R. C., Data Structures and Their Implementations, unpublished paper, University of Pennsylvania, December 1968.

Adams Associates Inc. Report AFCRL-68-0128, Development of Hierarchical Graphical Logical Entity Capabilities, by E. N. Chase and D. A. Westlake, 15 February 1968.

Cotton, I. W. and Greatorex, F. S., "Data Structures and Techniques for Remote Computer Graphics," AFIPS Conference Proceedings, Fall Joint Computer Conference, 1968, p. 533-544.

Evans, D. and Van Dam, A., "Data Structure Programming System," Proceedings of IFIP Congress 1968, p. 557-563.

Hurwitz, A., Citron, J. P. and Yeaton, J. B., "GRAF: Graphic Additions to FORTRAN," AFIPS Conference Proceedings, Spring Joint Computer Conference, 1967, p. 553-557.

Ross, D. T., "The AED Free Storage Package," Communications of the ACM, v. 10, no. 8, p. 481-492, August 1967.

Ross, D. T. and Rodriguez, J. E., "Theoretical Foundations for the Computer-Aided Design System," AFIPS Conference Proceedings, Spring Joint Computer Conference, 1963, p. 305-322.

Savitt, D. A. and Love, H. H., "ASP: A New Concept in Language and Machine Organization," AFIPS Conference Proceedings, Spring Joint Computer Conference, 1967, p. 87-102.

University of Illinois Coordinated Science Laboratory Report R-393, Cylinders: A Data Structure Concept Based on Rings, by P. Weston and S. M. Taylor, September 1968.

Van Dam, A. and Evans, D., "A Compact Data Structure for Storing, Retrieving and Manipulating Line Drawings," AFIPS Conference Proceedings, Spring Joint Computer Conference, 1967, p. 601-610.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	LT R. D. DeLaura, USNR (Code 52) Department of Electrical Engineering Naval Postgraduate School Monterey, California 93940	1
4.	LT L. H. Miller, Jr., USN 207 West 4th Street Burnet, Texas 78611	1

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1 ORIGINATING ACTIVITY (Corporate author)
Naval Postgraduate School
Monterey, California 93940

2a. REPORT SECURITY CLASSIFICATION
Unclassified

2b. GROUP

3 REPORT TITLE

A Graphics-Applications Data Structure for Medium Scale Computers

4 DESCRIPTIVE NOTES (Type of report and inclusive dates)
Master's Thesis; December 1970

5 AUTHOR(S) (First name, middle initial, last name)

Luke H. Miller, Jr.

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
December 1970	82	10

8a. CONTRACT OR GRANT NO.

b. PROJECT NO.

c.

d.

9a. ORIGINATOR'S REPORT NUMBER(S)

9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)

10. DISTRIBUTION STATEMENT
This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES

12. SPONSORING MILITARY ACTIVITY
Naval Postgraduate School
Monterey, California 93940

13. ABSTRACT

This paper documents the process by which a data structure for a Computer-Aided-Design system was selected and implemented on a medium scale computer. Included is a survey of the types of structures currently used in C.A.D. applications and a discussion of their capabilities and resource requirements.

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Computer-aided design						
SLIP						
Graphics data structures						